TAMPEREEN
AMMATTIKORKEAKOULU

JAVAFXLIBRARY

Robot Framework -kirjasto JavaFX-sovelluksille

Pasi Saikkonen

Opinnäytetyö
Toukokuu 2018
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

SAIKKONEN PASI:
JavaFXLibrary
Robot Framework -kirjasto JavaFX-sovelluksille

Opinnäytetyö 34 sivua, joista liitteitä 0 sivua
Toukokuu 2018

Tässä opinnäytetyössä kehitettiin JavaFX-sovellusten testaukseen soveltuva kirjasto Robot Framework -työkalulle. Tarve kirjastolle syntyi, kun toimeksiantajayritys Eficoden asiakkaan kehittämälle sovellukselle lähdettiin rakentamaan automatisoituja testejä. Valmista JavaFX-sovellusten testaamiseen sopivaa Robot Framework -kirjastoa ei ollut saatavilla, joten sellaista lähdettiin kehittämään TestFX-nimisen Java-kirjaston ympärille.

JavaFXLibraryn kehittäminen aloitettiin Eficodella keväällä 2017, ja sen versio 0.4.0 julkaistiin vuoden 2018 maaliskuussa Eficoden Github-sivuilla avoimena lähdekoodina. Julkaisuversio sisälsi 135 dokumentoitua Robot Framework -avainsanaa. Kirjastoa varten kehitettiin lisäksi useita pieniä JavaFX-sovelluksia ja Robot Framework -testejä, joilla kirjaston toimivuutta pystyttiin testaamaan kehityksen aikana.

Kirjastoa voidaan käyttää tavallisena Robot Framework -kirjastona, mutta se voidaan käynnistää myös etäkirjastotilaan. Etäkirjastona käytettynä testejä on mahdollista suorittaa myös Robot Frameworkin Python-versiolla. Siirtyminen versioiden välillä ei edellytä minkäänlaisia muutoksia testien toteutuksiin, vaan esimerkiksi Java-olioiden käsittely onnistuu myös Python-versiolla. Kirjasto sisältää kirjanpidon testeissä käytetyille Java-olioille ja avainsanoja olioiden käsittelyä varten.

Tuotoksena syntynyt kirjasto soveltuu erilaisten JavaFX-sovellusten testaukseen, ja sitä on jo käytetty menestyksekkäästi Eficoden asiakasyrityksessä. Kirjasto on avointa lähdekoodia, ja sitä voi käyttää kuka tahansa. Kirjaston kehitys jatkuu avoimen lähdekoodin käytäntöjen mukaisesti alkuperäisen kehitystiimin jatkaessa projektin ylläpitäjinä. Jatkokehitykseen soveltuvia aiheita ovat esimerkiksi tutoriaalisarjan tekeminen kirjaston ominaisuuksista sekä avainsanojen käytettävyyden tutkiminen ja kehittäminen.

Asiasanat: robot framework, javafx, ohjelmistotestaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

SAIKKONEN PASI:
JavaFXLibrary
Robot Framework Library for JavaFX Applications

Bachelor's thesis 34 pages, appendices 0 pages
May 2018

The goal of this thesis was to create a Robot Framework library which could be used to test JavaFX applications. The need for this kind of library rose when the client of Eficode was developing automated acceptance tests for their product, but could not find a suitable library for the purpose.

The development of JavaFXLibrary began at Eficode in spring 2017. Version 0.4.0 of the library was released as open source in March 2018. Release version contained 135 keywords.

JavaFXLibrary can be used both as a local or a remote library. Local use requires the Jython version of Robot Framework, as libraries written in Java usually do. When JavaFXLibrary is used as a remote library, tests can be executed with the regular Python version of Robot Framework as well. This makes the library very compatible, as it can be used with other test libraries that are created for different versions and implementations of the Python language.

JavaFXLibrary is an open source project, so it can be used by anybody and its development is open to everybody. The original development team will continue working with the library as project maintainers. Potential topics for further development are creating a tutorial series about the features of the library, and enhancing the usability of the keywords.

Key words: robot framework, javafx, software testing

SISÄLLYS

LYHENTEET JA TERMIT

API	Application Programming Interface, ohjelmointirajapinta
CSS	Cascading Style Sheets, tiedostomuoto tyyliohjeille
FXML	XML-pohjainen merkintäkieli käyttöliittymien määrittelyyn
GUI	Graphical User Interface, graafinen käyttöliittymä
Java	ohjelmointikieli
JavaFX	Javan käyttöliittymäkirjasto
Jython	Python-toteutus Javalle
MVC-arkkitehtuuri	suunnittelumalli
Python	ohjelmointikieli
Robot Framework	testiautomaatiokehys
TestFX	Java-kirjasto JavaFX-sovellusten testaukseen
W3C	WWW:n standardeja kehittävä ja ylläpitävä konsortio
XML	Extensible Markup Language, merkintäkielistandardi

1 JOHDANTO

Eficoden asiakasyrityksellä oli kehitteillä sovellus, jonka käyttöliittymän osia oli toteutettu JavaFX-tekniikalla. Sovellukselle haluttiin kehittää automatisoidut hyväksyntätestit, mutta testityökaluksi aiotulle Robot Frameworkille ei ollut vapaasti saatavilla kirjastoa, jolla sovellusta olisi voitu testata. Tämän opinnäytetyön aiheena oli tällaisen kirjaston tekeminen.

Kirjastoa kehitettiin Eficodella kolmen hengen tiimissä, johon kuuluivat lisäkseni Jukka Haavisto ja Sami Pesonen. Kehitystyön lisäksi Jukka ja Sami toimivat myös asiakasyrityksen päässä ottamassa kirjastoa käyttöön jo sen kehitysvaiheessa.

Projektin tavoitteena oli luoda avoimen lähdekoodin kirjasto, joka tarjoaa testaajille valmiita avainsanoja tyypillisiin käyttötapauksiin. Kirjastoa lähdettiin rakentamaan aluksi ainoastaan Jythonilla käytettäväksi Java-kirjastoksi, mutta projektin edetessä siitä päätettiin tehdä myös Python-yhteensopiva etäkirjasto.

Kirjastoa varten tehtiin useita pieniä JavaFX-testisovelluksia, joilla sen toimivuutta pystyttiin testaamaan kehityksen eri vaiheissa. Näitä testisovelluksia käyttäen kirjastolle luotiin Robot Framework- hyväksyntätestit, jotka toimisivat myöhemmin myös esimerkkeinä kirjaston käytöstä. Hyväksyntätestejä on käytetty myös kirjaston ominaisuuksien esittelyyn siitä kiinnostuneille.

Projekti siirtyi avoimen lähdekoodin vaiheeseen 23. maaliskuuta 2018, kun kirjaston versio 0.4.0 julkaistiin Eficoden Github-sivuilla. Kirjaston julkaisuversio sisälsi yhteensä 135 avainsanaa. Kehitystiimin jäsenet jatkavat kirjaston parissa projektin ylläpitäjinä.

2 ROBOT FRAMEWORK

2.1 Tausta

Robot Framework on avoimen lähdekoodin geneerinen testaustyökalu, joka sai alkunsa Pekka Klärckin diplomityöstä vuonna 2005. Sitä kehitettiin aluksi Nokia Networksilla, ja vuonna 2008 Robot Framework 2.0 julkaistiin avoimena lähdekoodina. (DEVOPS 2016: Robot Framework-työkalun pääkehittäjä… 2016.) Nykyisin projektia ylläpitää Robot Framework Foundation, johon myös Eficode kuuluu (Robot Framework Foundation n.d.).

2.2 Avainsanat

Robot Framework -testit koostuvat testivaiheita kuvaavista avainsanoista. Esimerkiksi sisäänkirjautumiseen liittyvässä testissä avainsanoja voisivat olla ”kirjoita käyttäjätunnus”, ”kirjoita salasana” ja ”paina login-painiketta”. Avainsanoja voidaan toteuttaa suoraan Robot Frameworkin omalla syntaksilla tai luomalla avainsanakirjastoja yhteensopivilla ohjelmointikielillä. Avainsana on käytännössä metodikutsu, joka sisältää testivaiheen toteutuksen. Avainsanoille voidaan antaa kutsun yhteydessä parametreja, ja ne voivat myös palauttaa arvoja tavallisten metodikutsujen tapaan (Robot Framework User Guide 2017b).

Testiä varten voidaan myös luoda korkeamman tason avainsanoja, jotka koostuvat matalampien tasojen avainsanoista. Yleisesti testitiedosto sisältää korkeimman tason testikohtaiset avainsanatoteutukset, joissa kutsutaan kirjastojen avainsanoja ja käsitellään testin muuttujia. Korkeamman tason avainsanoja käyttämällä testin kulku voidaan kuvata selkokielisesti myös henkilöille, jotka eivät välttämättä ymmärrä alemman tason avainsanakutsujen syntaksia. Avainsanoihin sovellettavaa abstraktiotasoa tulee kuitenkin punnita tapauskohtaisesti, sillä useammat avainsanatasot voivat esimerkiksi hankaloittaa testilokien lukemista. Aiemman esimerkin ”kirjoita käyttäjätunnus”-avainsana voisi esimerkiksi sisältää todellisuudessa kursorin siirtämisen käyttäjätunnus-kentän päälle, hiiren painikkeen painamisen ja näppäinpainallukset kullekin käyttäjätunnuksen kirjaimelle.

Vaikka alemman tason vaiheet sisällytetään korkean tason avainsanakutsuun, ne käsitellään edelleen myös yksittäisinä avainsanakutsuina niin testin suorituksessa kuin sen tuloksissakin. Tämän ansiosta alemman tason avainsanasuorituksen epäonnistuessa raportteihin pystytään kirjaamaan tarkkaan se kohta, joka aiheutti testin epäonnistumisen. Alemman tason avainsanan epäonnistuessa myös samaan testiin liittyvät korkeampien tasojen avainsanat merkitään epäonnistuneiksi.

2.3 Kirjastot

Matalan tason avainsanat sijaitsevat kirjastoiksi kutsutuissa moduuleissa. Standardikirjastoihin kuuluvat BuiltIn-, Collections-, DateTime-, Dialogs-, OperatingSystem-, Process-, Screenshot-, String-, Telnet- ja XML-kirjastot, jotka asennetaan aina osana Robot Frameworkia. Testeissä käytettäviä avainsanoja voidaan lisätä myös ulkoisilla kirjastoilla, joita on listattuna esimerkiksi Robot Frameworkin kotisivuilla.

Ulkoiset kirjastot vaativat erillisen asennuksen, jonka vaiheet vaihtelevat asennettavasta kirjastosta riippuen. Yksinkertaisimmillaan kirjasto on yksittäinen luokka, jonka sisältämät metodit toimivat suoraan avainsanoina. Kirjaston pohjana voidaan käyttää kolmea erilaista rajapintaa, jotka ovat static, dynamic ja hybrid. Niiden väliset erot muodostuvat avainsanojen toteutustavoissa. JavaFXLibraryn pohjalla on käytetty Javalib Corea, joka on Java-toteutus dynamic-rajapinnalle. Javalib Corea käsitellään tarkemmin JavaFXLibraryn avainsanoista kertovassa luvussa.

Testissä käytettävät kirjastot esitellään testitiedoston alussa Pythonin moduuli-importointia muistuttaen. Ainoastaan sisäänrakennetun BuiltIn-kirjaston sisältämät avainsanat ovat aina käytettävissä ilman erillistä kirjaston tuontia.

Kirjastoille on mahdollista antaa myös kustomoitu nimi niiden tuonnin yhteydessä lisäämällä kirjastoimportin perään *WITH NAME* ja haluttu nimi. Etäkirjastoja käytettäessä kirjasto määritellään nimen sijasta palvelimen osoitteella, esimerkiksi komennolla *Library | Remote | http://localhost:8270* otetaan käyttöön tietokoneen portissa 8270 palveleva kirjasto. Kuviossa 1 havainnollistetaan erilaisten kirjastojen määrittämistä testitiedoston alussa.

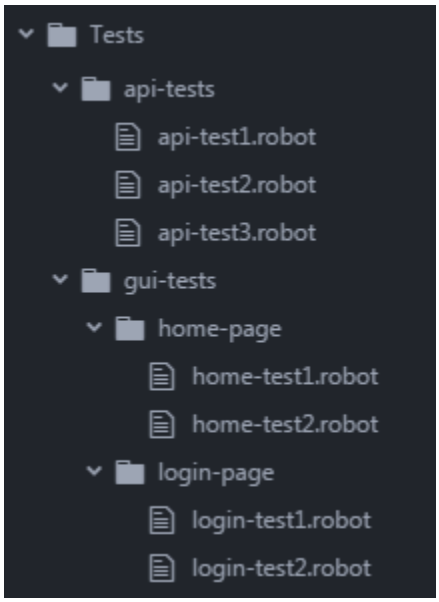
```
*** Settings ***
Library    Dialogs
Library    JavaFXLibrary
Library    Remote    http://127.0.0.1:8270    WITH NAME    MyLibrary
```

KUVIO 1. Erilaisia kirjastomäärittelyjä testin asetuksissa

2.4 Testin rakenne

2.4.1 Testikokoelma

Robot Framework -testi koostuu vähintään yhdestä testikokoelmasta, joka on testihierarkian korkeimman tason entiteetti. Jokainen testi on aina osa testikokoelmaa, joka voi pienimmillään olla yksittäinen testitiedosto. Testikokoelma voi myös sisältää muita testikokoelmia, jolloin testit voidaan jakaa testattavan sovelluksen rakenteen mukaisesti. Kuvassa 1 on havainnollistettu testien jakamista kokoelmiin erottamalla API- ja GUI-testit omiin kansioihinsa. Tämän lisäksi käyttöliittymätestit on jaoteltu vielä testattavan näkymän mukaisesti.



KUVA 1. Kokoelmiin jaetut testit

Testien jakaminen tämänkaltaisiin kokoelmiin on hyvä käytäntö, sillä se mahdollistaa yksittäisen sovellusalueen testaamisen ilman erillistä konfigurointia. Testikansioiden rakenne säilyy selkeänä ja helpompana ylläpitää, kun testit jaetaan sovelluksen kannalta järkeviin osiin. Hyvin suunniteltu rakenne parantaa myös testiraporttien luettavuutta, sillä testien tulokset jaetaan raporteissa kokoelmien mukaan. Raportissa listataan lisäksi kokoelmakohtaiset tilastot, joiden avulla mahdollisten ongelmien laajuudesta ja sijainnista saadaan tarkempi kuva.

2.4.2 Testitiedosto

Robot Framework tukee HTML-, TSV- ja reST-tiedostomuotoja, sekä tavallisia tekstitiedostoja. Versiosta 2.7.6 lähtien tekstimuotoiset testitiedostot on voitu nimetä myös käyttämällä robot-päätettä tavallisen txt-muodon sijaan. (Robot Framework User Guide 2017f.) Kaikissa tässä työssä esiintyvissä testiesimerkeissä käytetään tekstitiedostomuotoa, jota tulisikin Robot Framework User Guiden (2017f) mukaan käyttää aina, kun erityistä tarvetta muille tiedostomuodoille ei ole.

Testitiedosto sisältää aina vähintään testitapaukset, joiden lisäksi siinä voidaan määritellä mahdollisia asetuksia, muuttujia ja avainsanoja. Robot Framework tarjoaa useita erilaisia tapoja toteuttaa testi; testitapauksille voidaan esimerkiksi luoda mallipohja, jota käytetään useammassa testissä erilaisilla muuttujilla. Avainsanoja ja muuttujia voidaan myös määritellä erillisissä resurssitiedostoissa, joiden sisältö tuodaan kirjastojen tapaan saataville varsinaisen testin asetuksissa. Testikohtaisia avainsanoja ja resurssitiedostoja käyttämällä varsinaiset testit sisältävä tiedosto pysyy selkeämpänä ja testin kulku voidaan esittää selkokielellä myös henkilöille, jotka eivät ymmärrä resurssitiedostoissa käytettyä syntaksia.

2.4.3 Testitapaus

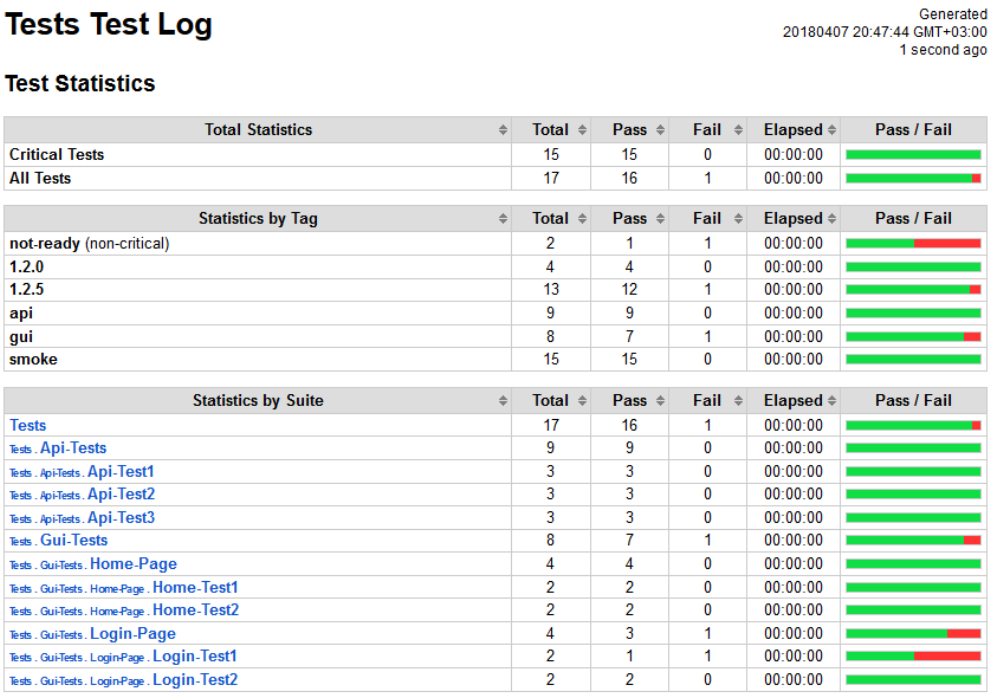
Testitapaus on testitiedoston sisältämä yksittäinen testi, jolla on aina vähintään nimi ja yksi suoritettava avainsana. Yleensä testi sisältää ennakkotilan asettamisen, testattavan tapahtuman sekä tuloksen vahvistamisen. Testin tulos on aina joko hyväksytty tai hylätty, muita vaihtoehtoja ei ole olemassa. Testin epäonnistuessa myös sen sisältävä testikoko-

elma merkitään epäonnistuneeksi, ellei kyseistä testiä ole merkitty epäkriittiseksi. Testitapauksille on mahdollista määritellä tunnisteita, joita käyttämällä voidaan rajata kuhunkin testiajoon mukaan otettavia tapauksia. Tämä on hyödyllistä esimerkiksi silloin, kun halutaan suorittaa eri kokoelmissa olevia, mutta toisiinsa jollain tasolla liittyviä testejä.

2.4.4 Testiraportit

Testiajon päätteeksi testitulokset tallennetaan XML-tiedostoon, jonka pohjalta generoidaan log- ja report-nimiset HTML-dokumentit. Log-tiedosto sisältää yksityiskohtaisemmat kuvaukset testien vaiheista, report-tiedoston ollessa pikemminkin testikokoelman tulosten yleiskuvaus (Robot Framework User Guide 2017c). Näissä dokumenteissa esitetään testien tulokset ja testitapausten vaiheet aikaleimoineen. Testilokin taso voidaan määrittää myös erikseen testin ajon yhteydessä, jolloin testien vaiheet kuvataan tavallista tarkemmin. Asetettavia lokitasoja on kuusi: none, fail, warn, info, debug ja trace (Robot Framework User Guide 2017d). Vakioasetuksena on info, jolloin lokiin kirjataan käytettyjen avainsanojen määrittelemät tulosteet.

Testiajon tulokset esitetään lokin alussa tunnisteiden ja testikokoelmien mukaan jaoteltuina (kuva 2). Yleiskatsauksen alta löytyy listaus suoritettujen testien vaiheista avainsanan tarkkuudella. Vaiheet esitetään puurakenteessa avainsanojen tason mukaan, testikokoelma juurisolmuna. Testin onnistuessa yksittäiset vaiheet ovat piilotettuina, kun taas testin epäonnistuessa loki aukeaa valmiiksi virheen aiheuttaneen avainsanan kohdalle helpottaen ongelman paikantamista.



Tests Test Log

Generated
20180407 20:47:44 GMT+03:00
1 second ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	15	15	0	00:00:00	
All Tests	17	16	1	00:00:00	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
not-ready (non-critical)	2	1	1	00:00:00	
1.2.0	4	4	0	00:00:00	
1.2.5	13	12	1	00:00:00	
api	9	9	0	00:00:00	
gui	8	7	1	00:00:00	
smoke	15	15	0	00:00:00	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Tests	17	16	1	00:00:00	
Tests . Api-Tests	9	9	0	00:00:00	
Tests . Api-Tests . Api-Test1	3	3	0	00:00:00	
Tests . Api-Tests . Api-Test2	3	3	0	00:00:00	
Tests . Api-Tests . Api-Test3	3	3	0	00:00:00	
Tests . Gui-Tests	8	7	1	00:00:00	
Tests . Gui-Tests . Home-Page	4	4	0	00:00:00	
Tests . Gui-Tests . Home-Page . Home-Test1	2	2	0	00:00:00	
Tests . Gui-Tests . Home-Page . Home-Test2	2	2	0	00:00:00	
Tests . Gui-Tests . Login-Page	4	3	1	00:00:00	
Tests . Gui-Tests . Login-Page . Login-Test1	2	1	1	00:00:00	
Tests . Gui-Tests . Login-Page . Login-Test2	2	2	0	00:00:00	

Kuva 2. Yleiskatsaus testiajon tuloksista

2.5 Testin suorittaminen

Testit suoritetaan yleensä komentoriviltä robot-skriptillä, mutta Robot Frameworkin asennustavasta riippuen testiajo voidaan käynnistää myös python- tai java-komentoa käyttämällä (Robot Framework User Guide 2017e). Käynnistyskomennolle annetaan parametrina vähintään testitapaukset sisältävän kokoelman polku. Komennossa voidaan määritellä myös asetuksia testiajolle, kokoelmasta voidaan esimerkiksi rajata suoritettavia testitapauksia nimen tai tunnisteen perusteella (Robot Framework User Guide 2017a). Asetuksilla pystytään rajaamaan esimerkiksi keskeneräiset tai keskeneräisiin ominaisuuksiin liittyvät testit pois testiajosta, tai ne voidaan merkitä ei-kriittisiksi.

Testin asetuksissa määritellään myös ulkoisten kirjastojen sijainti. Robot Framework käyttää kirjastotuontiin Python-tulkkia, jonka moduulipolussa sijaitsevat kirjastot ovat käytettävissä automaattisesti ilman erillistä konfigurointia. Esimerkiksi suoritettavan testin kanssa samassa hakemistossa sijaitsevaa testikirjastoa voidaan käyttää testitiedostoissa suoraan. Muualla sijaitsevat kirjastot voidaan lisätä Robot Frameworkin saataville lisäämällä ne käytetyn Python-tulkin polkuun, tai käyttämällä testiajon yhteydessä pythonpath-asetusta. Tällä asetuksella voidaan määritellä kirjastojen sijainnit myös Jythonilla ja IronPythonilla suoritettaville testeille (Robot Framework User Guide 2017a).

Robot Frameworkin JAR-versiota käytettäessä kirjastot lisätään Javan luokkapolkuun. Kirjastot voidaan määritellä myös JAR-versiota käytettäessä käynnistyskomennon yhteydessä Javan cp- tai classpath-asetuksella.

Testit voidaan suorittaa myös automatisoidusti osana jatkuvan integraation ja käyttöönoton prosessia. Työvaiheet riippuvat tietenkin käytetyistä työkaluista, mutta käytännössä automatisoituun suoritukseen lisätään sama komentorivikäsky, jota käytettäisiin manuaalisessakin testiajossa. Lisäksi esimerkiksi suositulle Jenkins-ohjelmistolle on saatavilla Robot Framework- lisäosa, joka helpottaa testien ja tulosten käsittelyä.

3 JAVAFX

3.1 Tausta

JavaFX on graafisten käyttöliittymien toteuttamiseen tarkoitettu kirjasto, jonka ensimmäinen versio julkaistiin vuonna 2008 (Vos, Chin, Gao, Weaver & Iverson 2017, 3). Se on kuulunut Java Standard Edition- ympäristöön versiosta 2.2 alkaen. JavaFX korvaa vanhemman Swing-kirjaston Javan standardikirjastona graafisten käyttöliittymien rakentamiseksi, mutta Swing aiotaan siitä huolimatta säilyttää Java Runtime Environmentin osana jatkossakin. (JavaFX Frequently Asked Questions n.d.)

3.2 JavaFX-ohjelman rakenne

Sovellusluokka

JavaFX-sovelluksen pääluokan on aina periydyttävä abstraktista Application-luokasta ja ylikuormitettava sen start-metodi (Chappell, Potts & Hildebrandt 2013, 8; Lowe 2015, 15). Application-luokka tarjoaa muutaman elinkaarimetodin, joiden avulla voidaan hallita esimerkiksi resurssien alustamista ja tuhoamista sovelluksen käynnistymisen ja sulkeutumisen yhteydessä. Sovellukselle ei tarvitse toteuttaa Javan main-metodia, jos se pakataan jar-tiedostoksi JavaFX Packager- työkalulla (Chappell ym. 2013, 8).

FXML- ja CSS-tiedostot

JavaFX:llä voidaan luoda graafisia käyttöliittymiä Swingin tapaan suoraan ohjelmakoodissa, tai käyttämällä erillisiä FXML-tiedostoja sovelluksen ulkoasun määrittelyyn. Näitä tapoja voidaan myös yhdistellä, esimerkiksi käyttöliittymän runko voidaan toteuttaa FXML-tiedostolla ja sisältö dynaamisesti Java-koodilla. JavaFX-komponenttien tyylit on mahdollista määritellä vielä erillisessä CSS-tiedostossa. JavaFX:n käyttämä CSS perustuu W3C:n CSS 2.1-versioon, sisältäen jonkin verran JavaFX:lle räätälöityjä lisäominaisuuksia.

FXML- ja CSS-tiedostojen käyttöä pidetään hyvänä käytäntönä (Hommel 2014, 13-17). FXML-tiedostojen käyttäminen käyttöliittymän määrittelyyn helpottaa MVC-arkkitehtuurin mukaisten sovellusten toteuttamista. Tyylimäärittelyjen siirtäminen CSS-tiedostoihin selkeyttää koodia ja parantaa sen ylläpidettävyyttä.

Käyttöliittymän osat

JavaFX-sovelluksen graafinen käyttöliittymä koostuu Stage-, Scene- ja Node-luokkien olioista. Stage on sovelluksen pääkomponentti, joka sisältää sovelluksen muut osat. (Vivien 2010, 48-49.) Käytännössä se on tietokoneen käyttöjärjestelmän ohjelmaa varten luoma ikkuna, jonka sisällä sovelluksen käyttöliittymä sijaitsee (Lowe 2015, 68). Stage-oliolle asetetaan Scene-instanssi, joka sisältää sovelluksen käyttöliittymän. Käyttöliittymä koostuu Node-luokasta periytyvistä komponenteista, jotka muodostavat puurakenteen (Vivien 2010, 49). Komponenttipuun juurisolmuna toimiva Node-objekti määritetään aina Scene-olion luonnin yhteydessä (Lowe 2015, 72).

Säikeet ja JavaFX

JavaFX sallii käyttöliittymän päivittämisen ainoastaan omassa JavaFX Application- säikeessään (Fedortsova 2012, 5). Komponentteja voidaan päivittää muista säikeistä Platform-luokan staattisella runLater-metodilla, jolle annetaan parametrina Runnable-rajapinnan toteuttava olio. JavaFX-komponentille suunnatut toimenpiteet määritellään parametrin run-metodin toteutuksessa, josta ne poimitaan JavaFX-säikeessä suoritettavien käskyjen jonoon. (Platform-luokan dokumentaatio n.d.)

JavaFX-säikeessä ei tulisi suorittaa aikaa vieviä prosesseja, sillä käyttöliittymä lakkaa vastaamasta käyttäjän syötteeseen säikeen ruuhkautuessa. Tällaisiin tilanteisiin suositellaan käytettäväksi JavaFX:n concurrent-pakettia, jonka avulla tehtävät voidaan suorittaa taustalla omissa säikeissään ja päivittää käyttöliittymää vain tarvittaessa. (Fedortsova 2012, 5.) Käytännössä on hyvä käyttää Platform-luokkaa yksinkertaisiin, suoraviivaisiin tilapäivityksiin ja concurrent-paketin luokkia, kun dataa haetaan esimerkiksi tietokannasta tai verkon yli.

4 JAVAFXLIBRARY

4.1 Tausta

Eficoden asiakasyrityksellä oli työn alla sovellus, jonka käyttöliittymän osia oli toteutettu JavaFX-komponenteilla. Tälle sovellukselle haluttiin tehdä automatisoidut hyväksyntätestit Robot Frameworkilla, mutta JavaFX-sovellusten testaamiseen soveltuvaa kirjastoa ei ollut tuolloin saatavilla. Asiaa tutkittaessa ilmeni myös, että tällaiselle kirjastolle löytyisi kysyntää. Tältä pohjalta JavaFXLibrarya lähdettiin kehittämään keväällä 2017.

4.2 Rakenne

JavaFXLibrary on TestFX-kehyksen ympärille rakennettu Robot Framework- kirjasto. Se on dynaaminen, Javalla toteutettu kirjasto, jonka perustana on käytetty Robot Frameworkin Javalib Core -kehyksen AnnotationLibrary-kirjaston pohjaa. Kirjastoon kuuluvat avainsanatoteutusten lisäksi myös Robot Framework- hyväksyntätestit ja niitä varten rakennetut JavaFX-testisovellukset. JavaFXLibrarya voidaan käyttää myös etäkirjastona.

4.2.1 TestFX

Kirjasto käyttää TestFX-ohjelmistokehystä JavaFX-sovellusten käsittelyyn. TestFX on avoimen lähdekoodin projekti, jonka tarkoituksena on mahdollistaa yksinkertaisten ja siistien testien tekeminen JavaFX:lle. JavaFXLibrarya lähdettiin alkuun kehittämään toteuttamalla TestFX:n rajapintojen tarjoamat ominaisuudet Robot Frameworkille.

TestFX käyttää testattavan sovelluksen ja käyttöjärjestelmän tilan manipulointiin kolmea erilaista ohjelmistorobottitoteutusta: omaa javafx-robottiaan, Javan awt-paketin Robot-luokkaa, sekä Sunin glass.ui-paketin Robot-luokkaa. Nämä toteutukset eroavat jonkin verran toisistaan, esimerkiksi javafx-robotti ei kykene toimimaan JavaFX-sovelluksen ulkopuolella, kun taas awt-robotti toimii käyttöjärjestelmän tasolla. TestFX pyrkii käyttämään ensisijaisesti omaa toteutustaan, mutta monet toiminnot vaativat operoimista JavaFX-ympäristön ulkopuolella, vaatien awt- ja glass-pakettien toteutusten käyttämistä.

Erilaisia robottitoteutuksia käytetään joillain alueilla myös limittäin, esimerkiksi kuvakaappauksien ottamiseen käytetään parametreista riippuen joko javafx- tai awt-robottia. Noodien kuvaamiseen käytetään javafx-robottia ja muun muassa näytön alueen kaappaamiseen awt-robottia.

Kun kirjastolle tehtiin kuvankaappauksiin liittyviä testejä, törmättiin tästä aiheutuvaan ongelmaan: kaksi samasta alueesta, mutta eri metodeilla otettua kuvaa eivät yllättäen olleetkaan yhtenäisiä. Niiden värien sävyt erosivat toisistaan hieman, ja toisessa näkyi suurennettuna reunanpehmennystä, kun toisen reunat olivat täydellisen terävät. Nämä erot johtuivat erilaisten robottitoteutusten käytöstä metodien välillä.

TestFX:n toteutus käyttää noodien kuvaamiseen JavaFX:n Node-luokan omaa snapshot-metodia. Se palauttaa tarkan kuvan, jossa värit ja niiden rajat ovat tismalleen samassa muodossa, kuin ne on kooditasolla määritetty. Tällä tavalla otettuihin kuviin ei tallennu myöskään kohteen lisäksi mitään muuta, vaan kuvattavan komponentin läpinäkyvien kohtien kohdalle piirtyy näkyvän taustan sijaan tyhjää. Awt-robotti taas ottaa näyttökaappauksen oikeasti, jolloin esimerkiksi käytetty väriprofiili ja näytön pikselitiheys vaikuttavat kuvan lopputulokseen. Tästä johtuen eri robottitoteutusten kuvat eivät ole välttämättä identtisiä, ja awt-robotilla otetut kuvat voivat erota toisistaan myös silloin, kun ne on otettu erilaisilla tietokoneilla tai näyttöasetuksilla.

4.2.2 Avainsanat

JavaFXLibraryn avainsanat on jaoteltu aihealueittain erillisiin Java-luokkiin, esimerkiksi kaikki näppäimistöön liittyvät avainsanat löytyvät KeyboardRobot-luokan alta. Jakamalla avainsanat useampaan tiedostoon kirjaston rakenne säilyy selkeämpänä ja koodi helpompana hallita, kun avainsanojen määrä kasvaa.

Kirjasto on toteutettu käyttämällä Javalib Coren AnnotationLibrary-kehystä, joka mahdollistaa dynaamisten Robot Framework- kirjastojen toteuttamisen tarjoamiensa Java-annotaatioiden avulla. Käytettävissä olevat annotaatiot ovat RobotKeywords, RobotKeyword, RobotKeywordOverload, ArgumentNames sekä Autowired.

RobotKeywords-annotaatiolla merkitään avainsanametodeja sisältävä luokka. Avainsanoja sisältävän luokan on oltava julkinen, siltä on löydyttävä vakiokonsturktori, ja sen tulee sijaita kirjaston avainsanojen sijainniksi määritetyllä polulla. Avainsanoja sisältävä luokka ei voi myöskään olla abstrakti. (Javalib Core Wiki 2013.)

Avainsanoina toimivat metodit merkitään RobotKeyword-annotaatiolla. Näiden metodien täytyy sijaita aina RobotKeywords-merkityn avainsanaluokan sisällä. (Javalib Core Wiki 2013.) Metodien nimet kääntyvät avainsanojen nimiksi niin, että camelcase-kirjoituksella eritellyt sanat jaetaan välilyönnein. Esimerkiksi *printHelloWorld*-niminen metodi kääntyy *Print Hello World*- avainsanaksi. Avainsanalle voidaan lisätä myös dokumentaatio kirjoittamalla se sulkeisiin annotaation perään (kuvio 2). Dokumentaatiot näkyvät testilokeissa ja niistä voidaan generoida kirjastodokumentaatio Javadocin tapaan.

```
@RobotKeyword("Prints Hello World!")
public void printHelloWorld() {
    System.out.println("Hello World!");
}
```

KUVIO 2. Dokumentoitu avainsanatoteutus

Avainsanat voivat ottaa vastaan parametreja, jotka merkitään ArgumentNames-annotaatiolla. Annotaatiolla määritetään argumenttien nimet, lukumäärä ja pakollisuus. Nimet lisätään sulkuihin annotaation perään avainsanadokumentaation tapaan. Argumenttien nimet kirjoitetaan aaltosulkujen sisään merkkijonomuodossa pilkuilla toisistaan erotettuina. Argumentti voidaan merkitä valinnaiseksi lisäämällä ’=’-merkki sen nimen perään. Jos parametrin tyyppi on Javan merkkijonotaulukko, se voidaan merkitä ottamaan vastaan vaihtelevan määrän argumentteja lisäämällä ’*’-merkki nimen eteen.

RobotKeywordOverload-annotaatiota käytetään, kun avainsana sisältää valinnaisia parametreja. Sillä merkitty metodi sisältää toteutuksen avainsanalle, kun valinnaista parametria ei ole annettu. Käytännössä tämä mahdollistaa vakioarvojen käyttämisen avainsanojen muuttujina (kuvio 3). Avainsanalla voi olla myös useampia ylikuormituksia, jos valinnaisia parametreja on enemmän kuin yksi.

```
@RobotKeyword
@ArgumentNames({"name", "amount="})
public void doStuff(String name, int amount) {
    callAnotherMethod(name, amount);
}

@RobotKeywordOverload
public void doStuff(String name) {
    callAnotherMethod(name, 4);
}
```

KUVIO 3. Amount-muuttujan arvona käytetään lukua 4 jos arvoa ei erikseen määritellä

Autowired-annotaation avulla muita avainsanaluokkia pystytään käyttämään luomatta niistä uusia olioita. Sillä merkittyyn muuttujaan injektoidaan kirjaston luonnin yhteydessä kyseisen avainsanaluokan instanssi, jolloin samaa oliota voidaan käyttää useammassa luokassa. Tämä on hyödyllistä varsinkin silloin, kun käytetyllä avainsanaluokalla on jokin tila, esimerkiksi tieto kursorin sijainnista näytöllä. Jos käytössä on yhtäaikaisesti useampi instanssi samasta luokasta, niiden tilat saattavat olla keskenään ristiriitaiset, jolloin eri avainsanaluokkien metodikutsut voivat toimia väärin ja johtaa odottamattomiin lopputuloksiin.

4.2.3 Testisovellukset ja hyväksyntätestit

Kirjastoa varten tehtiin useita pieniä JavaFX-sovelluksia, joilla avainsanojen oikea toiminta voitiin varmistaa. Näille sovelluksille luoduista testeistä muodostui ajan mittaan hyväksyntätestisarja, joka toimii myös esimerkkinä kirjaston käytöstä projektin siirryttyä avoimen lähdekoodin vaiheeseen.

Testisovellusten ja automaatiotestien käytöstä kirjaston kehityksen aikana on ollut paljon hyötyä. Niiden avulla on löydetty muutamia muutoin vaikeasti havaittavia ohjelmointivirheitä jo varhaisessa kehitysvaiheessa ja kehitetty kokonaan uusia, tarpeellisiksi havaittuja avainsanoja. Automatisoidun testisarjan avulla suurtenkin muutosten tekeminen on sujunut nopeasti, kun mahdolliset bugit on havaittu aikaisin ja kirjaston toimiminen muutosten jäljiltä on ollut helppoa varmistaa.

4.3 Kirjaston kehittäminen

Kirjastoa lähdettiin aluksi kehittämään toteuttamalla TestFX:n tarjoamia metodeja Robot Framework -yhteensopiviksi avainsanoiksi. Avainsanatoteutusten rinnalla kirjastolle rakennettiin räätälöityjä testisovelluksia, joiden avulla avainsanojen toimivuutta pystyttiin testaamaan. Avainsanojen toteutuksia ja niiden sanamuotoja muokattiin testisovelluksista saatujen kokemusten ja myöhemmin asiakasyrityksessä kerätyn palautteen perusteella.

Vaikka JavaFX-sovelluksille ei ollut olemassa Robot Framework -kirjastoa, Javan vanhemmalle Swing-käyttöliittymäkirjastolle sellainen löytyi. Tutkimme SwingLibrarya ja sen erillistä etäkirjastototeutusta RemoteSwingLibrarya määritellessämme yleisiä suuntaviivoja kirjaston avainsanatoteutuksille. SwingLibrary on ollut julkisesti saatavilla jo pitkään, sen ensimmäinen Githubissa julkaistu versio on päivätty vuodelle 2011. Javaan perustuvat käyttöliittymät tulevat tulevaisuudessa siirtymään Swingistä JavaFX:ään, jolloin kirjastomme yhtenäisyys SwingLibraryn kanssa tulisi helpottamaan testaajien siirtymistä tekniikoiden välillä. SwingLibrarysta päädyttiin kuitenkin lopulta ottamaan melko vähän mallia Swingin ja JavaFX:n erojen vuoksi. SwingLibraryssa avainsanat on myös määritelty yleisesti ottaen melko tarkalla tasolla, joka johtaisi kirjastomme tapauksessa valtavaan määrään avainsanoja ja täten heikentäisi kirjaston käytettävyyttä. SwingLibraryn avainsanoissa käytetty kieli noudattaa Robot Frameworkin hyviä käytäntöjä, johon pyrimme myös omaa kirjastoa kehittäessämme.

Altran julkaisi oman TestFX:ään pohjautuvan Robot Framework-kirjastonsa TestFXLibraryn Githubissa 27. helmikuuta 2018. Kirjastolla oli odotetusti jonkin verran päällekkäisyyksiä JavaFXLibraryn kanssa, käyttiväthän molemmat taustalla samaa testikehystä. Tutkiessamme Altranin kirjastoa huomasimme muutaman puutteen omassa toteutuksessamme, esimerkiksi taulukoiden käsittelyyn ei tarjottu tuolloin juurikaan valmiita avainsanoja. TestFXLibrary vaikuttaa ottaneen paljon vaikutteita SwingLibrarysta sisältäen yksityiskohtaisempia avainsanoja, kuten TextArea-objektin tekstiarvon hakemisen. Vertailun vuoksi omassa toteutuksessamme päädyimme käyttämään yleiskäyttöisempiä avainsanoja, joille voidaan antaa mikä tahansa Java-objekti.

TestFXLibrarya testatessamme ilmeni, etteivät kaikki sen avainsanat toimineet aina odotetulla tavalla. Esimerkiksi taulukon solun arvoa haettaessa arvon sijaan palautui Java-objekti merkkijonomuodossa, jota kirjastolla ei voinut käsitellä. TestFXLibrary on myös

tarjolla ainoastaan Jythonilla käytettävänä versiona, eikä se siitä huolimatta tukenut Java-olioiden käsittelyä testien puolelta. Kirjasto vaikutti kokonaisuutena hieman keskeneräiseltä, ja se erosi toteutuksestamme lopulta niin merkittävästi, että koimme omalle ratkaisullemme olevan yhä tilausta.

Kirjaston käyttäjän kannalta avainsanat ja niiden dokumentaatio ovat keskeisessä roolissa. JavaFXLibrary sisälsi julkaisunsa aikaan 135 avainsanaa, joka on huomattavasti vähemmän kuin SwingLibraryn 189 ja TestFXLibraryn 186 avainsanaa. Ennen kirjaston julkaisua avainsanoja yhdisteltiin mahdollisuuksien mukaan ja niistä tehtiin yleisesti geneerisempiä. Tästä johtuen käyttäjän ei tarvitse tietää, minkä luokan objektia testissä käsittelee, sillä kirjasto pystyy hoitamaan tyyppitarkastukset itse. Useille avainsanoille on asetettu myös vakioarvoja, esimerkiksi klikatessa hiiri siirtyy automaattisesti suorinta reittiä kohteeseen. Käyttäjä voi halutessaan muuttaa kursorin liikerataa antamalla avainsanalle valinnaisen parametrin.

Avainsanojen määrän rajoittaminen helpottaa kirjaston käyttöä varsinkin alkuvaiheessa, kun käyttäjän on helpompi löytää itselleen tarpeellisia avainsanoja. Käytettävyyden kannalta on tärkeää löytää tasapaino avainsanojen määrän ja kirjaston ominaisuuksien välillä. Päädyimme ennen kirjaston julkaisua poistamaan jonkin verran avainsanoja, joiden käyttötapausten ajattelimme olevan harvinaisia todellisessa käytössä. Kirjaston hyväksyntätesteissä nämä poistetut avainsanat korvattiin Javan metodikutsuilla, joten kirjastolla voidaan tarvittaessa esimerkiksi tarkistaa kahden neliön mahdollinen päällekkäisyys tai sovellusikkunan reunojen leveys. Avainsanoja määritellessämme pyrimme siihen, että kirjastosta löytyisi valmiit toteutukset yleisimpiin käyttötarkoituksiin, tarjoten lisäksi mahdollisuuden toteuttaa testikohtaisia avainsanoja olioiden käsittelyn ja metodikutsujen avulla.

Avainsanojen muotoilu on äärimmäisen tärkeää, jotta niiden tarkoitus selviäisi käyttäjälle jo nimen perusteella. Robot Framework- kirjastoissa käytetään yleensä sisäänrakennettujen kirjastojen mukaista kieltä, esimerkiksi arvon tarkastamiseen käytetyt avainsanat kirjoitetaan yleisesti ”x should be y”-muodossa. Yhtenäinen kieli parantaa testitiedoston luettavuutta, kun käytössä on useita eri kirjastoja. Useille ohjelmointiympäristöille on myös saatavilla Robot Framework- lisäosia, jotka tarjoavat kirjastojen avainsanoja testiä kirjoi-

tettaessa. Editori saattaa esimerkiksi listata kaikki should-alkuiset avainsanat, joista käyttäjä voi sitten valita tarvitsemansa. Kirjastodokumentaatiota ei tarvitse juurikaan edes lukea, kun avainsanat ovat selkeästi muotoiltuja ja käytetty kieli yhtenäistä.

JavaFXLibraryn avainsanojen nimeäminen oli melko haasteellista. Käytimme aluksi pitkälti TestFX:n metodeista ja luokista johdettuja nimiä, mutta ne eivät tuntuneet aina kovinkaan intuitiivisilta tai kuvaavilta. Päädyimme poistamaan useita TestFX:n sisäisiin luokkiin liittyviä avainsanoja, joiden emme kokeneet tuovan mitään lisäarvoa käyttäjälle. Muokkasimme myös jäljelle jääneiden avainsanojen sanamuotoja kuvaavammiksi, ja yhdistelimme niitä mahdollisuuksien mukaan.

4.4 Kirjaston ominaisuuksia

4.4.1 Komponenttien etsiminen CSS-selektoreilla

JavaFX:n Node-luokka sisältää lookup- ja lookupAll-metodit, joiden avulla voidaan etsiä sovelluksen käyttöliittymäkomponentteja. Lookup-metodi palauttaa ensimmäisen selektorilla löytyvän node-objektin, ja lookupAll kaikki kuvausta vastaavat komponentit. Komponenttien etsimiseen käytetään merkkijonomuotoisia CSS-selektoreita, joilla etsittävää noodia voidaan kuvata esimerkiksi id-arvolla tai tyyliluokilla. Javan bugista johtuen pseudoselektorit eivät kuitenkaan toimi useampaa komponenttia haettaessa.

TestFX:n avulla komponentteja voidaan etsiä suoraan koko sovelluksesta, ja hakuja voidaan tehdä CSS-selektoreiden lisäksi myös noodien tekstiarvoilla sekä Matcher- ja Predicate-luokilla (TestFX Wiki 2016). TestFX:n haussa luokat kuvataan CSS-selektoreissa aina pistemuodossa, esimerkiksi hierarkia ”*VBox HBox Button*” merkitään TestFX:n haussa *”.vBox .hBox .button*”. Tämä rajoitus johtuu mahdollisuudesta käyttää tekstiarvoja haussa: jos pistettä tyyliluokan edessä ei ole, TestFX yrittää etsiä komponenttia, jonka tekstinä on ”*VBox HBox Button*” sen sijaan, että etsisi vastaavaa tyyliluokkahierarkiaa. JavaFXLibrary yhdistelee sekä TestFX:n että JavaFX:n tarjoamia hakutoimintoja, joten komponentteja on mahdollista etsiä monipuolisilla hakulauseilla eri kohdista käyttöliittymän puurakennetta.

4.4.2 Komponenttien ja niiden tilojen odottaminen

Kirjasto sisältää avainsanoja, joilla voidaan odottaa komponentin syntyä tai sen tilan muutosta. Tällainen tarve syntyy esimerkiksi silloin, kun käyttöliittymän sisältöä haetaan tietokannasta eivätkä tulokset saavu välittömästi. Ilman avainsanan käyttöä kyseinen testi epäonnistuu siihen, ettei haettua komponenttia ole vielä olemassa.

Komponentin odottamisen lisäksi kirjastosta löytyy avainsanat latauspalkin valmistumisen, sekä node-olion visible- ja enabled-arvojen muutosten odottamista varten. Odotusaika voidaan määritellä tapauskohtaisesti avainsanakutsun yhteydessä. Vakiona odotusajat vaihtelevat avainsanasta riippuen viiden ja kahdenkymmenen sekunnin välillä.

4.4.3 Klikkausten varmistus

Varsinkin kirjaston kehityksen alkuvaiheissa hyväksyntätestejä suoritettaessa saattoi käydä niin, että kursori siirtyi testattavan sovelluksen ulkopuolelle ja klikkaili tai raahasi työpöydän kuvakkeita. Tämä johtui huonosti suunnitelluista testeistä, joissa ei esimerkiksi koordinaatteja käsiteltäessä huomioitu riittävästi erilaisia resoluutioita ja näyttöasetuksia. Tällainen virhe tapahtuu kuitenkin helposti testejä kirjoittaessa, ja se saattaa aiheuttaa todellisia vaaratilanteita: Robot Framework voi vahingossa poistaa tiedostoja tai muuttaa tietokoneen asetuksia. Tämän tapahtuessa testin keskeyttäminen voi olla myös todella hankalaa, sillä Robot Framework kontrolloi samaan aikaan hiirtä ja näppäimistöä, kun käyttäjä yrittää keskeyttää käynnissä olevaa testiajoa.

Vahinkojen estämiseksi kirjastolle kehitettiin varmistin, joka tarkastaa kursorin sijainnin ennen hiiren painikkeiden painamista. Kursorin täytyy sijaita testattavan sovelluksen sisältämien ikkunoiden alueella tai klikkausta ei suoriteta ja testi keskeytetään. Tarvittaessa klikkausten varmistus voidaan kytkeä avainsanalla pois päältä, jolloin kirjastolla voidaan toimia jälleen myös käyttöjärjestelmätasolla. Näppäimistön käyttäminen testattavan sovelluksen ulkopuolella vaatii push- press- ja release-avainsanojen käyttämistä, sillä write-avainsanalla on mahdollista kirjoittaa ainoastaan JavaFX-komponentin sisään.

4.4.4 Kuvien vertailu

Kirjaston varhaiset versiot sisälsivät myös kuvantunnistukseen liittyviä avainsanoja, joilla voitiin esimerkiksi etsiä kuvake näytöltä ja klikata sitä. Tunnistus perustui tarkasteltavan kuvan jakamiseen kohdekuvan mukaan määriteltyihin alueisiin ja näiden värikeskiarvojen vertailuun. Kuvantunnistus päädyttiin kuitenkin poistamaan kirjastosta ennen sen julkaisua, sillä siitä oli haastavaa saada tarpeeksi luotettava.

Kuvantunnistusta varten Robot Frameworkille löytyy esimerkiksi ImageHorizonLibrary-kirjasto, jota voidaan käyttää vastaavasti kuvakkeen sijainnin paikantamiseen näytöllä. JavaFXLibrary sisältää yhä muutaman avainsanan yksinkertaiseen kuvien vertailuun, mutta kaikki kohteiden tunnistukseen liittyvät luokat ja avainsanat on poistettu. Yleisesti testejä tehtäessä pyritään kuitenkin välttämään kuvantunnistuksen käyttöä, sillä siitä aiheutuu herkästi ongelmia etenkin erilaisissa ympäristöissä ajettuna.

4.5 Remote Library -rajapinta

4.5.1 Toimintaperiaate

JavaFXLibrarya on mahdollista käyttää myös etäkirjastona. Robot Frameworkiin sisältyy Remote library-rajapinta, joka mahdollistaa kirjastojen käyttämisen palvelimen kautta. Etäkirjastoa käyttämällä testejä voidaan ajaa ilman paikallista kirjastoasennusta.

Etäkirjasto ja Robot Framework kommunikoivat XML-RPC-protokollan yli, joten kirjasto voidaan toteuttaa millä tahansa protokollaa tukevalla kielellä (Klärck 2017). Usealle ohjelmointikielelle on saatavilla valmiita etäkirjastoja varten tehtyjä palvelintoteutuksia, joiden avulla kirjaston muuttaminen etäkirjastoksi on yksinkertaista. JavaFXLibrary käyttää tähän tarkoitukseen Javalla kirjoitettua jrobotremoteserveriä.

4.5.2 Jrobotremoteserver

Javalla toteutetuista Robot Framework -kirjastoista voidaan tehdä etäkirjastoja jrobotremoteserverin avulla. JavaFXLibraryssa palvelimen käynnistys on sisällytettynä kirjaston main-metodiin, joten etäkirjasto voidaan käynnistää suorittamalla kirjaston sisältävä jar-paketti.

Etäkirjastoa käyttämällä päästään eroon Jython-riippuvuudesta, sillä kirjastoa käyttävät testit voidaan suorittaa Python-pohjaisella robot-toteutuksella. Tämä mahdollistaa myös Jythonin kanssa yhteensopimattomien kirjastojen käyttämisen samoissa testeissä. Lisäksi testien suorittaminen nopeutuu, kun Robot Frameworkin ei tarvitse odotella Javan virtuaalikonetta testiajon käynnistyksen yhteydessä.

Etäkirjastoon siirtymiseen liittyy kuitenkin JavaFXLibraryn tapauksessa myös rajoituksia. Osassa kirjaston hyväksyntätesteistä käsiteltiin Java-olioita suoraan testitiedostojen puolella, esimerkiksi metodikutsuja avuksi käyttäen. Tämä oli mahdollista, koska testit suoritettiin Java-pohjaisen Jythonin päällä. Etäkirjastoon siirryttäessä oliot kuitenkin lakkaavat liikkumasta testin ja kirjaston välillä, ja testien puolelle palautuvat objektit saapuvatkin merkkijonomuodossa. Tästä johtuen Java-metodien kutsuminen testien puolelta muuttuu mahdottomaksi.

4.5.3 ObjectMap

Olioiden käsittelyyn liittyvät rajoitukset vaikuttivat kirjaston käyttöön merkittävästi. Jokaiselle käyttötapaukselle ei ole mahdollista luoda omaa avainsanaansa, jolloin kyky etsiä objekti käyttöliittymästä ja kutsua sen metodeja helpotti testien tekemistä todella paljon. Ilman suoraa olion käsittelemistä joidenkin testien tulosten varmistaminen muuttuisi käytännössä mahdottomaksi. JavaFX-sovellus voi sisältää paljon sovelluskohtaisia luokkia, joita voidaan tukea vain tietylle yhteiselle tasolle asti, esimerkiksi Node-luokasta periytyville olioille.

Vaikka jrobotremoteserveriä käytettäessä oliot palautuvatkin testeihin merkkijonomuodossa, osaa se myös tulkata joitain Javan tietotyyppejä Pythonille ja toisinpäin. Javan primitiiviset tietotyypit long-tyyppiä lukuun ottamatta toimivat suoraan, mutta short- ja

byte-tyyppiset luvut muuttuvat int-tyyppisiksi palatessaan testien puolelta. Perustietotyyppien wrapper-luokat, kuten Double ja Integer toimivat myös automaattisesti. Edellä mainittujen lisäksi vielä listat, taulukot ja mapit ovat yhteensopivia, mutta kaikki muut luokat muutetaan aina merkkijonomuotoon.

Kyky käsitellä olioita testien puolella koettiin tärkeäksi ominaisuudeksi, joten kirjastolle kehitettiin sitä varten objectMap-niminen kirjanpito. Se on Javan HashMap, johon tallennetaan jokainen testille palautettava olio, joka ei ole suoraan testeille yhteensopivassa muodossa. Mappiin lisätään avaimeksi Java-olion hashcodesta johdettu merkkijono ja arvoksi viittaus kyseiseen olioon. Testin puolelle palautetaan avaimena toimiva merkkijono.

Jokaisen avainsanan kutsun yhteydessä mahdolliset parametrit tarkistetaan kirjaston puolella, ja avaimina toimivat merkkijonot korvataan niiden avulla objectMapista löydetyillä olioilla. Näin testin puolella syntyy vaikutelma siitä, että käsiteltävät muuttujat olisivat oikeita Java-objekteja, vaikka ne todellisuudessa ovat vain olioihin viittaavia merkkijonoja (kuvio 4). Koska objektien haku kirjanpidosta tapahtuu automaattisesti aina avainsanan käytön yhteydessä, valmiita avainsanatoteutuksia ei jouduttu muuttamaan juurikaan etäkirjastoon siirryttäessä, vaan alkuperäinen koodi säilyi enimmäkseen yhteensopivana.

```
${node}     Find    .button
Click On    ${node}
```

KUVIO 4. Node-muuttuja käyttäytyy Java-olion tavoin, vaikka onkin merkkijono

Kirjanpidon arveltiin käyvän raskaaksi suuria sovelluksia testattaessa, joten objektien haun yhteyteen kehitettiin tarkistuksia mapin koon minimoimiseksi. Jos palautettava tyyppi on yhteensopiva jrobotremoteserverin tukemien tyyppimuunnosten kanssa, sitä ei lisätä kirjanpitoon ollenkaan, vaan sen annetaan liikkua testin ja kirjaston välillä sellaisenaan. Jos lisättävä objekti löytyy jo valmiiksi kirjanpidosta, siitä ei myöskään lisätä uutta merkintää, vaan testin puolelle palautetaan aiemmin luotu avain. Näin kirjanpitoon ei synny turhia duplikaatteja, vaan jokainen testattavan sovelluksen olio voi esiintyä siellä ainoastaan kerran.

ObjectMap voidaan myös tyhjentää avainsanaa käyttämällä testien puolelta. Testin aikana haettu olio on testitiedostossa käytettävissä ainoastaan sen testitapauksen ajan, jossa se on esitelty, joten oliota ei tarvitse enää testin vaihtuessa säilyttää kirjaston kirjanpidossa. Poikkeuksen tähän tekevät suite-tason muuttujat, jotka ovat käytettävissä koko testitiedoston alueella. Toisaalta mappia on myös tarpeetonta tyhjentää testien välissä, jos sama olio haetaan jokaisessa testissä erikseen. Kirjanpidon tyhjentävä avainsana voidaan lisätä tarpeen mukaan joko *Suite Teardown*- tai *Test Teardown*- asetuksen alle, jolloin kirjanpito tyhjennetään aina erillisten testitiedostojen tai yksittäisten testien välillä.

Kirjastolle tehtiin kirjanpidon lisäämisen myötä myös erilliset avainsanat Javan metodikutsujen käyttöä ja olioiden attribuuttien hakemista varten. Avainsanat on toteutettu hyödyntämällä Javan geneerisiä ominaisuuksia, joten ne tukevat myös sovelluskohtaisia luokkia. Metodien käyttäminen toimii käytännössä samalla tavoin kuin aiemmin Jython-pohjaista kirjastoa käytettäessä (kuvio 5).

```
${label}    Find    .label
${text}     Call Object Method    ${label}    getText
```

KUVIO 5. Label-olion getText-metodin kutsu testin puolelta

4.6 Kirjaston asentaminen

JavaFXLibraryn voi ladata JAR-pakettina projektin Github-sivulta. Halutessaan käyttäjä voi myös kopioida sivulta kirjaston lähdekoodin ja kääntää kirjaston itse käyttämällä Maven-työkalua. Komento *mvn package* paketoi kirjaston JAR-tiedostoon, joka vastaa Github-sivuilla ladattaviksi tarjottuja versioita. Paketointiin voidaan käyttää myös *mvn verify* -komentoa, joka ajaa kirjaston paketoinnin yhteydessä sen hyväksyntätestit.

Erillistä kirjastoasennusta JAR-paketin lisäksi ei tarvita. Paikallisesti Jythonilla suoritettuna kirjasto täytyy kuitenkin vielä lisätä Robot Frameworkin moduulipolkuun. Java-kirjastoissa tähän tarkoitukseen käytetään yleensä Javan luokkapolkua, joka voidaan määrittää myös testien käynnistämisen yhteydessä. Luokkapolkuun lisätään JAR-paketin sijainti, jonka jälkeen kirjastoa voidaan käyttää testeissä normaalisti. Luokkapolku voidaan määrittää myös osana testiajon käynnistävää komentoa käyttämällä Javan cp- tai classpath-asetusta. Etäkirjastona käytettynä polkumäärittelyitä ei tarvitse asettaa. Etäkirjasto

käynnistetään suorittamalla JAR-paketti, jonka jälkeen kirjasto on käytettävissä asetetussa portissa. Jos porttia ei erikseen määritellä käynnistyksen yhteydessä, käytetään Robot Frameworkin etäkirjastoille rekisteröityä vakioporttia 8270. Kun kirjastopalvelin on käynnissä, sitä voidaan käyttää testeissä määrittelemällä sen osoite testitiedostossa kirjaston tuonnin yhteydessä.

4.7 Kirjaston käyttäminen

Testien kannalta ainoa ero paikallisen ja etänä käytetyn kirjaston välillä on etäkirjaston osoitteen määrittäminen testin asetuksissa. Kirjaston avainsanat käyttäytyvät täysin samalla tavalla riippumatta siitä, miten kirjastoa käytetään. Java-olioiden sijasta kirjasto palauttaa testien puolelle merkkijonoja myös paikallisesti Jythonilla ajettuna, joten kirjaston käyttötapaa voidaan vaihtaa ilman minkäänlaisia muutoksia testitoteutuksiin.

JavaFXLibrary vaatii, että testattava sovellus käynnistetään sen avainsanaa käyttämällä. Kirjaston julkaisuversio ei vielä pysty löytämään valmiiksi avoinna olevaa JavaFX-sovellusta, mutta kyseinen ominaisuus tullaan mahdollisesti lisäämään kirjastolle tulevaisuudessa. Sovelluksen käynnistyksen yhteydessä suoritetaan myös kirjaston taustalla käytetyn TestFX:n vaatima konfigurointi. Testattava sovellus voidaan käynnistää antamalla avainsanalle parametrina JAR-paketti tai sovelluksen pääluokka.

5 TULOKSET JA JATKOKEHITYS

Opinnäytetyön tavoitteena oli kehittää JavaFX-sovellusten testaamiseen soveltuva Robot Framework -kirjasto, joka tarjoaa valmiita avainsanoja yleisiin hyväksyntätestauksen käyttötapauksiin. Tarve tällaiselle kirjastolle syntyi, kun Eficoden asiakasyrityksen kehittämälle sovellukselle haluttiin luoda automatisoidut hyväksyntätestit, eikä tarkoitukseen soveltuvaa kirjastoa löytynyt.

Tuotoksena syntyneen kirjaston versio 0.4.0 julkaistiin maaliskuussa 2018 avoimena lähdekoodina Eficoden Github-sivuilla. Työn voidaan katsoa saavuttaneen tavoitteensa, sillä kirjastoa on jo käytetty menestyksekkäästi asiakasyrityksen sovelluksen testaukseen. Kirjasto sisältää graafisten sovellusten testaamiseen suunniteltuja helppokäyttöisiä avainsanoja, joiden avulla voidaan esimerkiksi simuloida käyttäjän hiirellä ja näppäimistöllä antamaa syötettä.

Valmiiden avainsanaratkaisujen lisäksi kirjasto mahdollistaa myös Java-olioiden metodien kutsumisen suoraan testien puolelta. Näiden avainsanojen avulla testaajan on mahdollista manipuloida testattavan sovelluksen tilaa ja suorittaa tarkistuksia, joille ei ole mahdollista luoda valmiita avainsanoja. Tämä on hyödyllistä varsinkin silloin, kun sovelluksella on omia käyttöliittymäkomponenttitoteutuksia, joiden arvoja halutaan tarkistaa.

JavaFXLibraryn merkittävin etu on sen etäkirjastotuki. Kirjastoa lähdettiin aluksi kehittämään Jython-pohjaiseksi, mutta asiakasyritykselle toteutettujen testien pohjalta syntyi ajatus lisätä kirjastoon tuki etäkäytölle. Robot Framework -kirjastot ovat yleisesti siirtymässä Pythonin 3-versioon, eikä vielä ole varmaa, koska Jythonista julkaistaan Python3-yhteensopiva versio. Etäkirjastona käytettynä JavaFXLibrarya voidaan käyttää myös Jythonin kanssa yhteensopimattomien kirjastojen kanssa samoissa testeissä. ObjectMapin ansiosta alkuperäisen Jython-pohjaisen toteutuksen ja lopullisen etäkirjaston välillä ei ole käyttäjän kannalta mitään eroa.

Kirjaston avainsanatoteutuksia tullaan varmasti vielä muokkaamaan jossain määrin, mutta kokonaisuuden kannalta radikaaleja muutoksia tuskin enää tehdään. Yhtenä esimerkkinä kehittämistä vaativista avainsanoista voidaan käyttää metodikutsuihin liittyviä

toteutuksia. Tällä hetkellä parametreja vastaanottavien metodien kutsuminen testien puolelta on melko kömpelöä, sillä avainsanalle täytyy antaa kutsun yhteydessä sekä parametrit että tiedot näiden tietotyypeistä. Kummatkin argumentit vaaditaan myös valmiiksi listamuodossa, joka vaatii kahden listamuuttujan luontia Robot Frameworkin sisäänrakennettuja avainsanoja käyttämällä. Optimitilanteessa avainsanalle voitaisiin antaa parametreja vaihteleva määrä, ja kirjasto pystyisi määrittämään parametrien tietotyypit ja luokat automaattisesti. Tämän ominaisuuden lisäämistä yritettiin jo aiemmin, mutta metodikutsujen käyttöön ja tyyppimuunnoksiin liittyy useita jrobotremoteserveristä ja Javalib Coresta aiheutuvia rajoituksia, joiden takia muutoksen tekeminen on haastavaa.

Avainsanojen lisäksi muita keskeisiä jatkokehitysaiheita voisivat olla esimerkiksi jo käynnissä olevaan JavaFX-sovellukseen kiinnittyminen kirjaston avulla, tai perusteellinen tutoriaalisarja, jossa kirjaston keskeisiä ominaisuuksia käydään läpi mahdollisimman yksinkertaisin esimerkein. Kirjaston omat hyväksyntätestit soveltuvat aloittelijalle esimerkeiksi lopulta melko huonosti, sillä ne on tehty ensisijaisesti kirjaston toimintojen testausta varten. Testejä on tästä johtuen valtavasti ja niiden testitapaukset ovat toisinaan melko monimutkaisia, joten yksinkertaisemmista esimerkeistä olisi varmasti apua kirjaston käyttöä aloiteltaessa. Avainsanadokumentaatiosta löytyy toki esimerkkejä useimpien avainsanojen käytöstä, mutta harjoittelu tiettyä testisovellusta vasten ja valmiita esimerkkitapauksia seuraten olisi todennäköisesti opettavaisempaa.

Jatkokehitysaiheita tulee esiin varmasti vielä jatkossakin, kun kirjastoa käytettäessä löydetään uusia ohjelmointivirheitä, ongelmia käytettävyydessä tai puutteita toiminnoissa. Githubin kautta palautteen, ideoiden ja koodikorjaustenkin lähettäminen on helppoa. Kirjastolla on myös oma javafxlibrary-niminen kanava Robot Frameworkin Slack-keskustelualueella, joka on tarkoitettu kirjaston kehitykseen liittyvään keskusteluun.

6 POHDINTA

Kirjasto saavutti alkuperäisen tavoitteensa, ja sen julkaisuversio oli lopulta ominaisuuksiltaan huomattavasti laajempi kuin alun perin oli suunniteltu. Kirjastolle pystyttiin tekemään kesken kehityksen hyvinkin radikaaleja muutoksia, jotka saatiin vietyä läpi erittäin nopeasti. Kehitystiimi oli pieni, ja kaikki sen jäsenet osallistuivat suurempien muutosten yhteydessä pidettyihin katselmointeihin, joissa sovittiin jatkosta ja keskusteltiin esiin tulleista ongelmista ja muista kirjastoon tai sen ympäristöön liittyvistä havainnoista.

Kirjaston testisovellusten ja asiakasyrityksen testien perusteella tehtyjen huomioiden pohjalta päädyttiin tekemään suuria muutoksia niin avainsanatoteutuksiin kuin kirjaston arkkitehtuuriinkin. Muutosten ansiosta kirjaston avainsanat ovat helppokäyttöisempiä ja selvemmin muotoiltuja. Testisovelluksista oli huomattavasti hyötyä myös suurempien muutosten läpiviennissä, sillä niiden avulla löydettiin muutamia erittäin vaikeasti huomattavia ohjelmointivirheitä. Hieman ironisestikin kirjaston yksikkötestaaminen jäi kuitenkin hyvin ohueksi, ja ongelmaan alettiin kiinnittää kunnolla huomiota vasta kun Robot Framework -testien ajoaika oli venynyt useiden minuuttien pituiseksi.

JavaFXLibrary on äärimmäisen yhteensopiva muiden Robot Framework -kirjastojen kanssa, sillä sitä voidaan käyttää Pythonin eri versioille ja toteutuksille kirjoitettujen kirjastojen kanssa. Testitapauksiin ei tarvitse tehdä muutoksia Robot Frameworkin versiosta toiseen siirryttäessä, joten versioiden välillä vaihtaminen on helppoa. Python-yhteensopivuuden ansiosta kirjaston käyttäjän ei tarvitse myöskään murehtia Jythonin epävarmasta tulevaisuudesta, kun kirjasto ei vaadi sen käyttämistä.

JavaFXLibraryn pohjaksi valittu TestFX aiheutti aluksi hieman ongelmia ohjelmointivirheidensä vuoksi. TestFX:n kehitys on kuitenkin jatkunut melko tasaisesti, ja sille on julkaistu JavaFXLibraryn kehityksen aloittamisen jälkeen seitsemän päivitystä, jotka sisältävät useita virhekorjauksia. Koska kyseessä on myös avoimen lähdekoodin projekti, olemme pystyneet itsekin lähettämään korjauksia havaitsemiimme ongelmiin.

Vaikka kirjastoa lähdettiinkin alun perin kehittämään Eficoden asiakasyrityksen tarpeisiin, siitä on nyt avoimena lähdekoodina hyötyä myös kaikille muille Robot Frameworkin käyttäjille. Myös Eficoden muut asiakasyritykset ovat olleet kiinnostuneita kirjastosta.

Avoimen lähdekoodin periaatteiden mukaisesti kuka tahansa voi käyttää kirjastoa ja osallistua sen kehitykseen. JavaFXLibrary on listattuna Robot Frameworkin kotisivuilla muiden yleisten kirjastojen kanssa, joten se on helposti löydettävissä. Kirjastosta julkaistiin myös kirjoitus Eficoden blogissa, ja tarkoituksena on järjestää Robot Framework meetup -tapahtumia, joissa kirjaston ominaisuuksia ja sen käyttöä käsitellään tarkemmin.

LÄHTEET

Chappell, G., Potts, J. & Hildebrandt, N. 2013. Getting Started with JavaFX. Luettu 7.3.2018. https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.pdf

Gudehus, B. 2016. TestFX Wiki. Julkaistu 3.3.2016. Luettu 12.3.2018. https://github.com/TestFX/TestFX/wiki/Queries

Eficode. 2016. DEVOPS 2016: Robot Framework-työkalun pääkehittäjä Pekka Klärck nousee lavalle. Julkaistu 21.3.2016. Luettu 5.3.2018. https://www.eficode.com/blogi/blogi/devops-2016-robot-framework-tyokalun-paakehittaja-pekka-klarck-nousee-lavalle

Fedortsova, I. 2012. Concurrency in JavaFX. Luettu 7.3.2018. https://docs.oracle.com/javafx/2/threads/jfxpub-threads.pdf

Hommel, S. 2014. Implementing JavaFX Best Practices. Luettu 7.3.2018. https://docs.oracle.com/javafx/2/best_practices/jfxpub-best_practices.pdf

Javalib Core Wiki. 2013. AnnotationLibrary. Julkaistu 20.9.2011. Päivitetty 10.10.2013. Luettu 6.3.2018. https://github.com/robotframework/JavalibCore/wiki/AnnotationLibrary#introduction

Klärck, P. 2017. Remote library interface. Julkaistu 27.1.2017. Luettu 5.3.2018. https://github.com/robotframework/RemoteInterface#introduction

Lowe, D. 2015. Javafx for dummies. 1. painos. Hoboken: John Wiley & Sons, Inc.

Oracle. N.d. JavaFX CSS Reference Guide. Luettu 7.3.2018. https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html#intro

Oracle. N.d. JavaFX Frequently Asked Questions. Luettu 5.3.2018. http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html

Oracle, N.d. Platform-luokan dokumentaatio. Luettu 7.3.2018. https://docs.oracle.com/javase/8/javafx/api/javafx/application/Platform.html#runLater-java.lang.Runnable-

Robot Framework Foundation. N.d. Robot Framework Foundation - Maintaining and promoting Robot Framework. Luettu 5.3.2018. http://robotframework.org/foundation/

Robot Framework User Guide. 2017a. Configuring execution. Julkaistu 13.2.2017. Luettu 2.5.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#configuring-execution

Robot Framework User Guide. 2017b. Creating user keywords. Julkaistu 13.2.2017. Luettu 10.5.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#creating-user-keywords

Robot Framework User Guide. 2017c. Different output files. Julkaistu 13.2.2017. Luettu 15.4.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#different-output-files

Robot Framework User Guide. 2017d. Log levels. Julkaistu 13.2.2017. Luettu 15.4.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#log-levels

Robot Framework User Guide. 2017e. Starting test execution. Julkaistu 13.2.2017. Luettu 2.5.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#starting-test-execution

Robot Framework User Guide. 2017f. Supported file formats. Julkaistu 13.2.2017. Luettu 6.3.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#supported-file-formats

Vivien, V. 2010. JavaFX 1.2 application development cookbook: over 80 recipes to create rich Internet applications with many exciting features. 1. painos. Birmingham: Packt Publishing Ltd.

Vos, J., Chin, S., Gao W., Weaver J. & Iverson D. 2017. Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients. 4. painos. New York: Apress.